# *Measurement and Modeling for Resource Prediction and Control in Heterogeneous Active Networks*

**Virginie Galtier, Yannick Carlinet, Kevin L. Mills,
Stefan Leigh, and Andrew Rukhin**

**DARPA Active Networks PI Meeting**

**June 5, 2001**

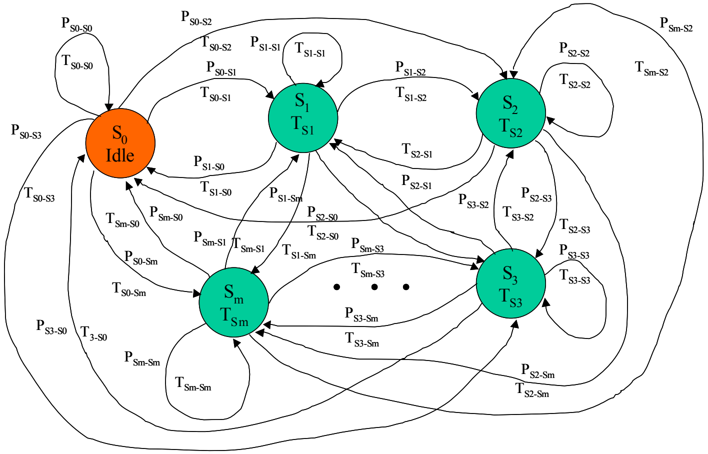
# *Outline of Presentation*

- Project Quad Chart
- Why is the problem important?
- Thumbnail: How have we tried to solve the problem?
- What have we done in the past 12 months?
  - Jun 00 to Dec 00: Demonstrate the application of our work to predict and control CPU usage in active applications (together with GE, Magician, and AVNMP)
  - Jan 01 to Jun 01: Turn the demonstrations into accurately measured, controlled experiments, confirming results, and writing papers and a Ph.D. dissertation
- Summary & What's next? (1) Future Research (Address Failures) (2) Prepare code and documentation for release on the project web site (3) Develop a white-box model

# Measurement and Modeling for Resource Prediction and Control in Heterogeneous Active Networks

NIST
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

## “How Much CPU Time?”



## Goal

Devise and validate a means to express the CPU time requirements of a mobile-code application in a form that can be meaningfully interpreted among heterogeneous nodes in an active network.

## Technical Approach

- Propose abstract mathematical models for active network nodes and active applications.
- Validate the abstract models against measurements from real active nodes and applications.
- Prototype validated mechanisms in a node operating system and evaluate them in live operations.

## Projected Impact

- Effective policy enforcement provides security and reliability essential for well-engineered active nets.
- An independent CPU metric permits resource requirements to be expressed across a heterogeneous set of active nodes to support:
    - node-level policy enforcement,
    - static path-level network admission control,
    - dynamic path-level QoS routing.

## FY01 Accomplishments

- Demonstrated the application of our work to predictive estimation and control of resource requirements for an Active Application. (collaboration with GE research)
- Based on encouraging results from the demonstration, modified Magician and AVNMP to make more precise and accurate measurements, and then conducted controlled experiments to confirm the demonstration.
- Published two papers on the application of our work.

# *Growing Population of Mobile Programs on Heterogeneous Platforms*

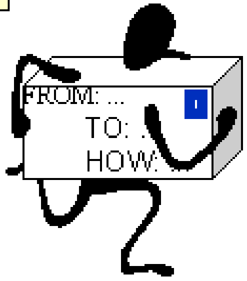
SCRIPTING ENGINES & LANGUAGES

Python

vbscript
jscript

APPLETS & SERVLETS

ACTIVE



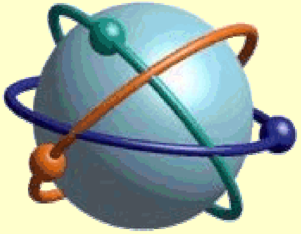
NETWORKS

**dlls, dlls, and more dlls**

Microsoft

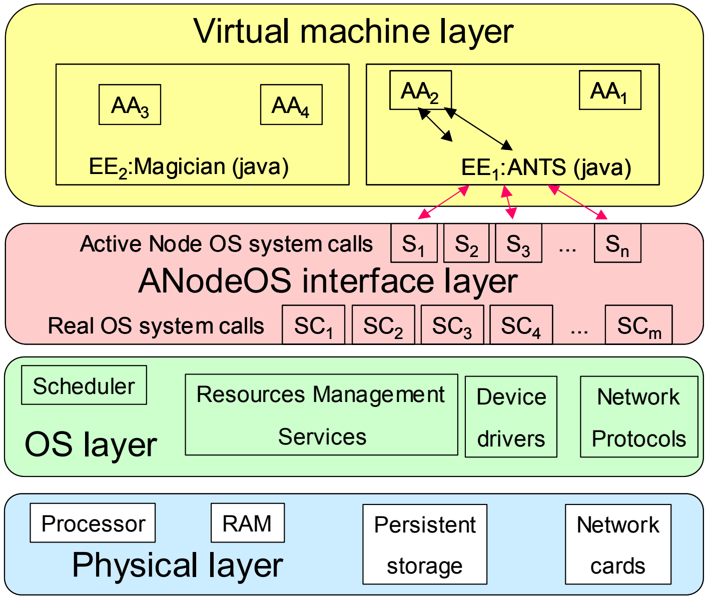
C#

MOBILE AGENTS



AgentSpace

# Sources of Variability



ANETS ARCHITECTURE

## VARIABILITY IN EXECUTION ENVIRONMENT

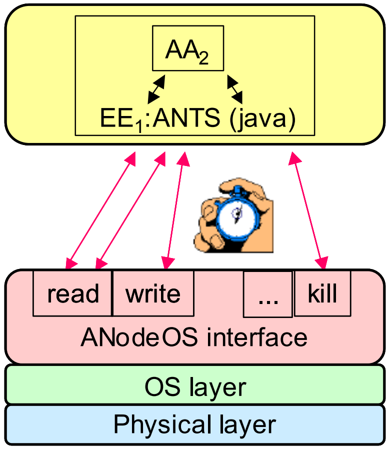
| Trait | Blue | Black | Green |
|---|---|---|---|
| CPU Speed | 450 MHz | 333 MHz | 199 MHz |
| Processor | Pentium II | Pentium II | PentiumPro |
| Memory | 128 MB | 128 MB | 64 MB |
| OS | Linux 2.2.7 | Linux 2.2.7 | Linux 2.2.7 |
| JVM | jdk 1.1.6 | jdk 1.1.6 | jdk 1.1.6 |
| Benchmark<br>Avg. CPU us<br>Avg. PCCs | <br>534<br>240,269 | <br>479<br>159,412 | <br>843<br>167,830 |

| | Blue | | Black | | Green | |
|---|---|---|---|---|---|---|
| System Call | pcc | us | pcc | us | pcc | us |
| read | 19,321 | 43 | 12,362 | 37 | 12,606 | 63 |
| write | 22,609 | 50 | 14,394 | 43 | 12,362 | 62 |
| socketcall | 27,066 | 60 | 17,591 | 53 | 14,560 | 73 |
| stat | 22,800 | 51 | 14,731 | 44 | 12,042 | 61 |

## VARIABILITY IN SYSTEM CALLS

# Our Approach in Thumbnail

**Monitor at System Calls in Active Node OS** → **Generate Execution Trace** → **Generate Active Application Model** →



...
begin, user (4 cc), read (20 cc), user (18 cc), write(56 cc), user (5 cc), end

begin, user (2 cc), read (21 cc), user (18 cc), □ kill (6 cc), user (8 cc), end

begin, user (2 cc), read (15 cc), user (8 cc), kill (5 cc), user (9 cc), end

begin, user (5 cc), read (20 cc), user (18 cc), write(53 cc), user (5 cc), end

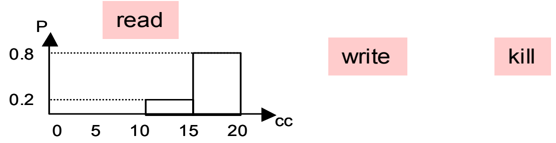
begin, user (2 cc), read (18 cc), user (17 cc), kill (20 cc), user (8 cc), end
...

***Trace is a series of system calls and transitions stamped with CPU time use***

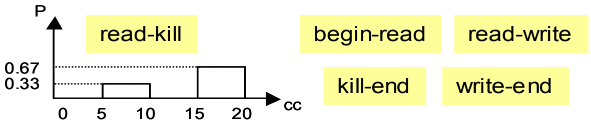
**Scenario A:**
sequence = “read-write”,
probability = 2/5

**Scenario B:**
sequence = “read-kill”,
probability = 3/5

**Distributions of CPU time in system calls :**

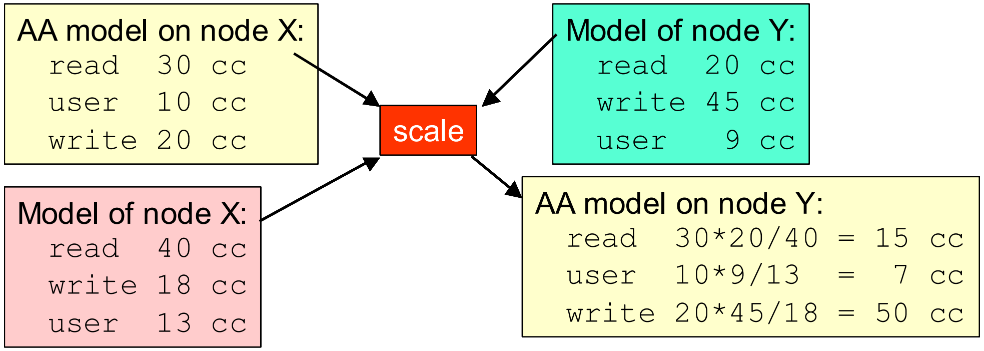


**Distributions of CPU time between system calls :**



→ **Simulate Model with Monte Carlo Experiment** → **Statistically Compare Simulation Results against Measured Data** → ***Scaling AA Models***

| | | 100 bins-20000 reps | |
|---|---|---|---|
| EE | AA | Mean | Avg. High Per. |
| ANTS | Ping | 0.86 | 0.9 |
| | Mcast | 0.40 | 1.9 |
| Magician | Ping | 0.44 | 33 |
| | Route | 0.73 | 13 |



AA model on node X:
```
read   30 cc
user   10 cc
write  20 cc
```

Model of node X:
```
read   40 cc
write  18 cc
user   13 cc
```

scale

Model of node Y:
```
read   20 cc
write  45 cc
user    9 cc
```

AA model on node Y:
```
read   30*20/40 = 15 cc
user   10*9/13  =  7 cc
write  20*45/18 = 50 cc
```

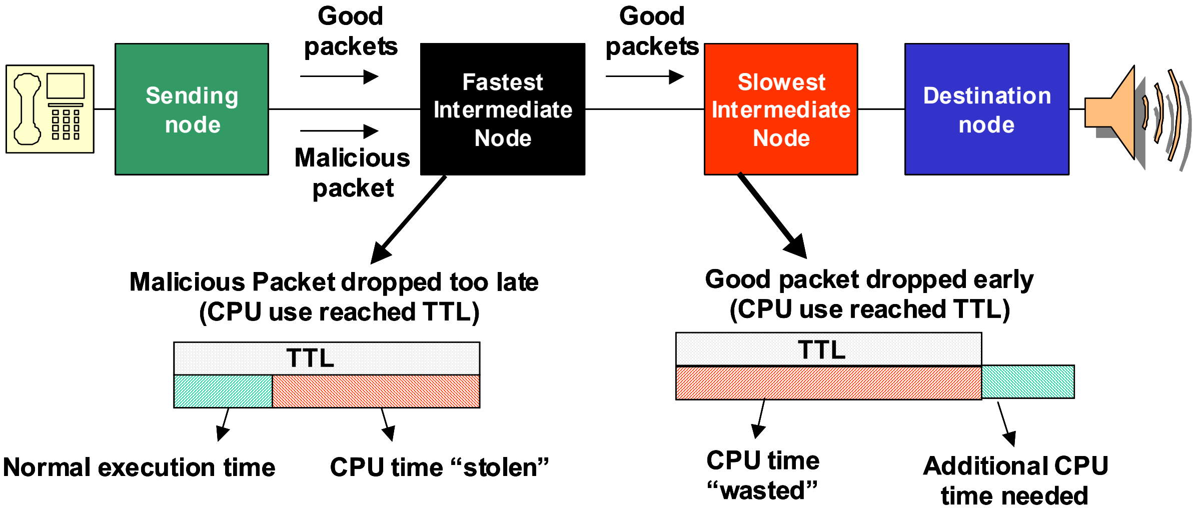
# *What have we done in the past 12 months?*

- Jun 00 to Dec 00: Demonstrated the application of our work to predict and control CPU usage in active applications (together with GE, Magician, and AVNMP)

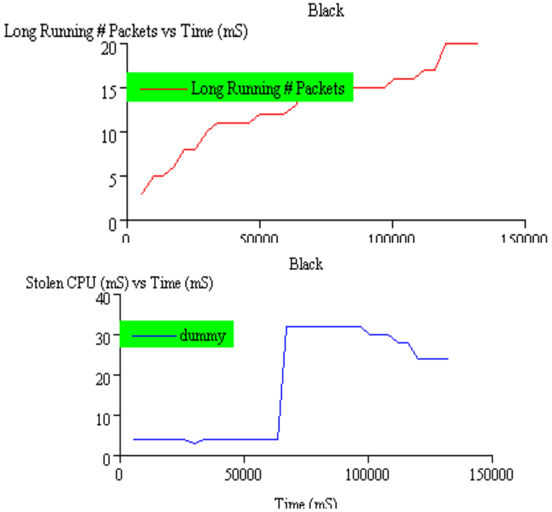
- Jan 01 to Jun 01: Turned the demonstrations into accurately measured, controlled experiments, confirming results, and writing papers and a Ph.D. dissertation

# Control Demo Revisited

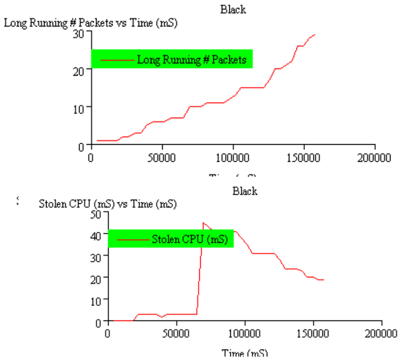
**Policy 1:** Use CPU time-to-live set to fixed value per packet
**Policy 2:** Use a CPU usage model, but scaled naively based solely on CPU speed
**Policy 3:** Use a well-scaled NIST CPU usage model



## Naïve Scaling



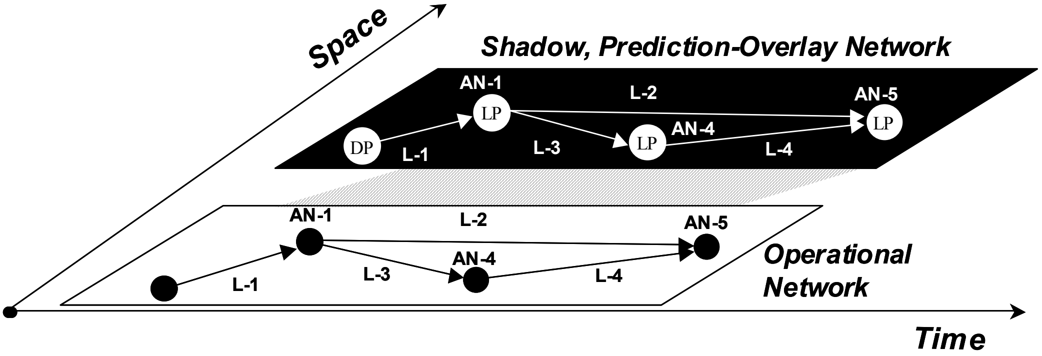
## High Fidelity

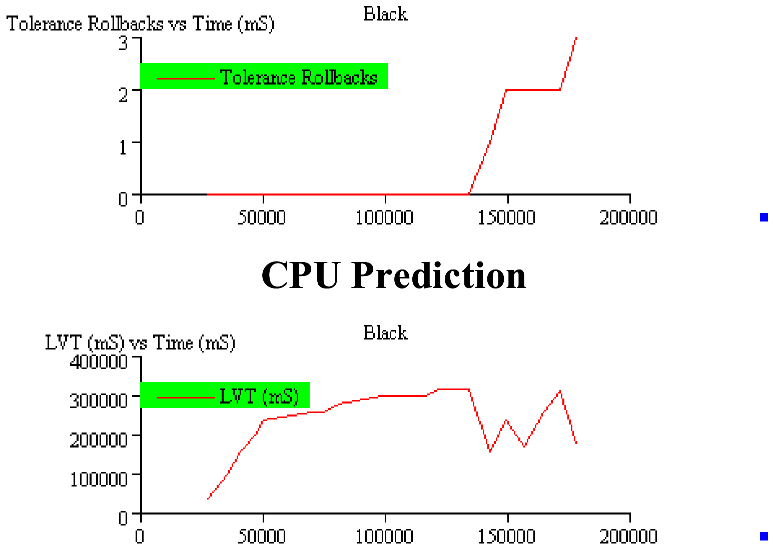


# Prediction Demo Revisited

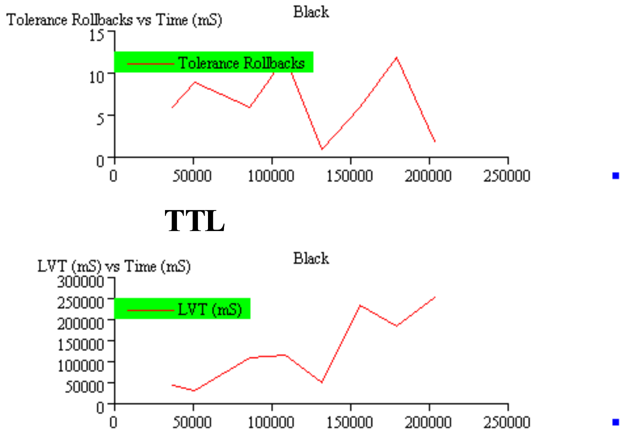
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

**With the NIST CPU usage model integrated, AVNMP requires fewer rollbacks**
**And so AVNMP can predict CPU usage in the network further into the future**

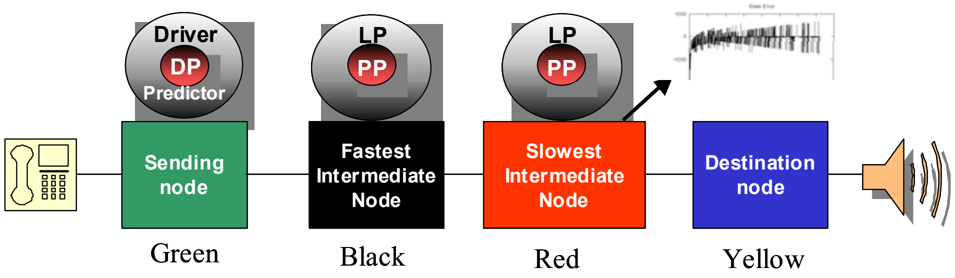




CPU Prediction



TTL



# *From Demo to Experiment*

- Completed **true** integration of NIST LINUX kernel measurement code with Magician and AVNMP control and prediction code (in Java)

- Modified AVNMP MIB to distinguish between CPU vs. packet stimulated rollbacks (and to prevent periodic resetting of values)

- Recalibrated demonstration nodes (and evaluated the calibrations and our models using selected Magician AAs – ping, route, activeAudio – new results given on next slide)

- Ran the two demonstrations again, but this time as controlled experiments (new results given on subsequent slides)

- Wrote two papers describing the control and prediction experiments

- Completed draft dissertation (Virginie Galtier)

# Evaluating Scaled AA Models

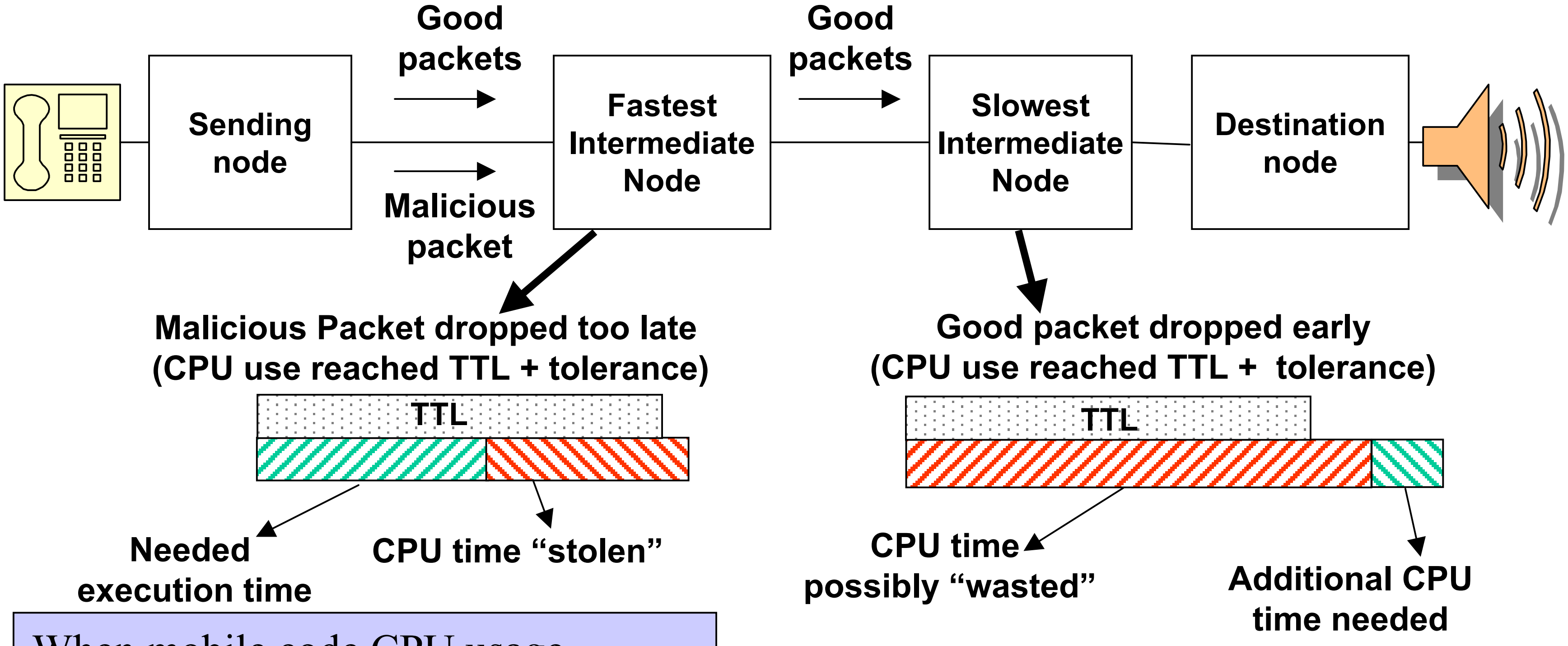
***Prediction Error Measured when Scaling Application Models between Selected Pairs of Nodes vs. Scaling with Processor Speeds Alone (MAGICIAN EE)***

## NEW RESULTS -- MAY 2001

| | | | Scaling with model | | Scaling with processors speeds | |
|---|---|---|---|---|---|---|
| AA | Node X | Node Y | Mean | Avg. High Perc. | Mean | Avg. High Perc. |
| Ping | K | B | 18 | 12 | 49 | 32 |
| | R | G | 21 | 32 | 74 | 72 |
| | R | K | 3 | 14 | 18 | 16 |
| | Y | K | 8 | 18 | 84 | 81 |
| | G | Y | 4 | 18 | 193 | 160 |
| Route | K | Y | 16 | 22 | 341 | 404 |
| | Y | R | 2 | 14 | 76 | 75 |
| | K | B | 6 | 13 | 13 | 30 |
| | G | K | 13 | 11 | 46 | 52 |
| | Y | G | 2 | 21 | 57 | 58 |
| Audio | Y | B | 11 | 27 | 85 | 83 |
| | K | Y | 13 | 14 | 400 | 399 |
| | G | Y | 9 | 10 | 80 | 143 |
| | G | B | 9 | 17 | 74 | 58 |
| | Y | K | 7 | 12 | 80 | 80 |

# *Experiment #1: Control Execution of Mobile Code*

Goals: (1) Show reduced CPU usage by terminating malicious packets earlier ***AND***
(2) Show fewer terminations of good packets


Good packets
Good packets
Sending node
Fastest Intermediate Node
Slowest Intermediate Node
Destination node
Malicious packet
Malicious Packet dropped too late
(CPU use reached TTL + tolerance)
Good packet dropped early
(CPU use reached TTL + tolerance)
TTL
TTL
Needed execution time
CPU time “stolen”
CPU time possibly “wasted”
Additional CPU time needed

When mobile code CPU usage controlled with fixed allocation or TTL, malicious or “buggy” mobile programs can “steal” substantial CPU cycles, especially on fast nodes

When mobile code CPU usage controlled with fixed allocation or TTL, correctly coded mobile programs can be terminated too soon on slow nodes, wasting substantial CPU cycles

# CPU Control: Experiment Results

## Summary of Results

| Metric | Fixed TTL Model | NIST CPU Model |
|---|---|---|
| Estimated Average CPU Utilization (ms/packet)<br>Fast Intermediate Node | 8.15 | 7.63<br>(6.3% estimated CPU time saved) |
| Good Packets Killed (all nodes) | 72 (3%) | 10 (0.4%) |

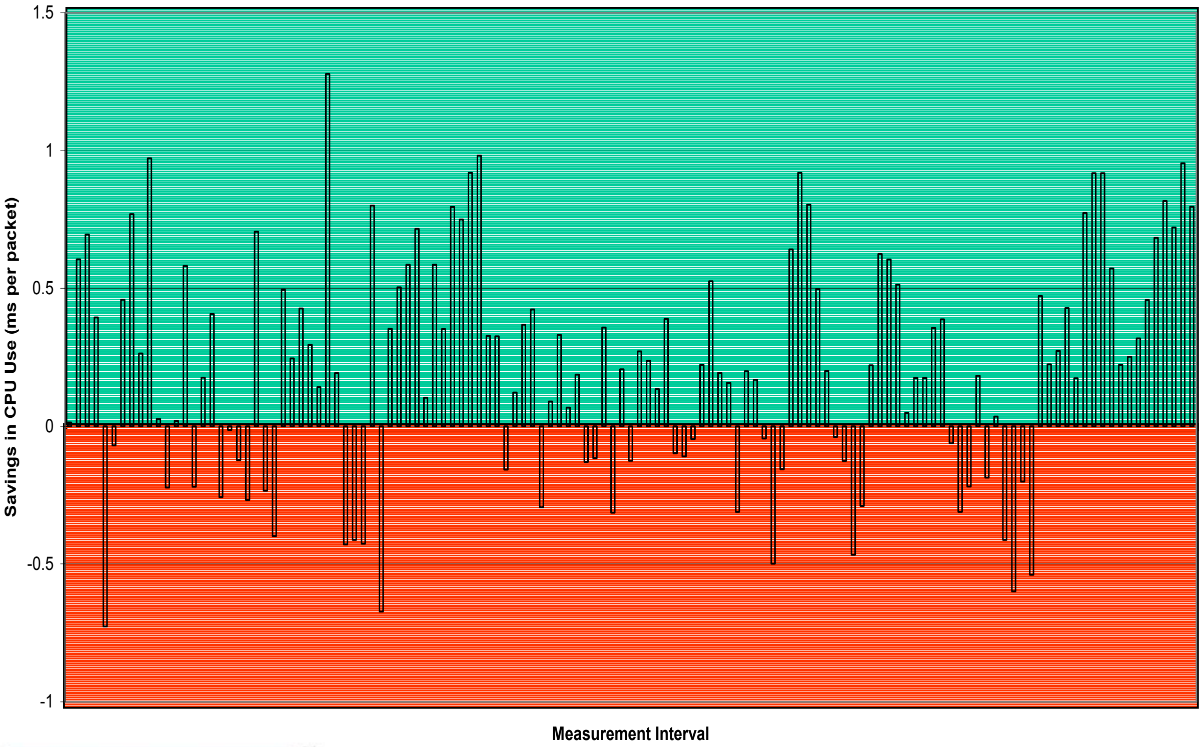
## Results vs. Prediction

| Predicted vs. Measured | |
|---|---|
| Predicted CPU time saved (ms/packet) [ANALYSIS] | 0.5 |
| Estimated CPU time saved (ms/packet) [EXPERIMENT] | 0.52 |

## Experiment Parameters

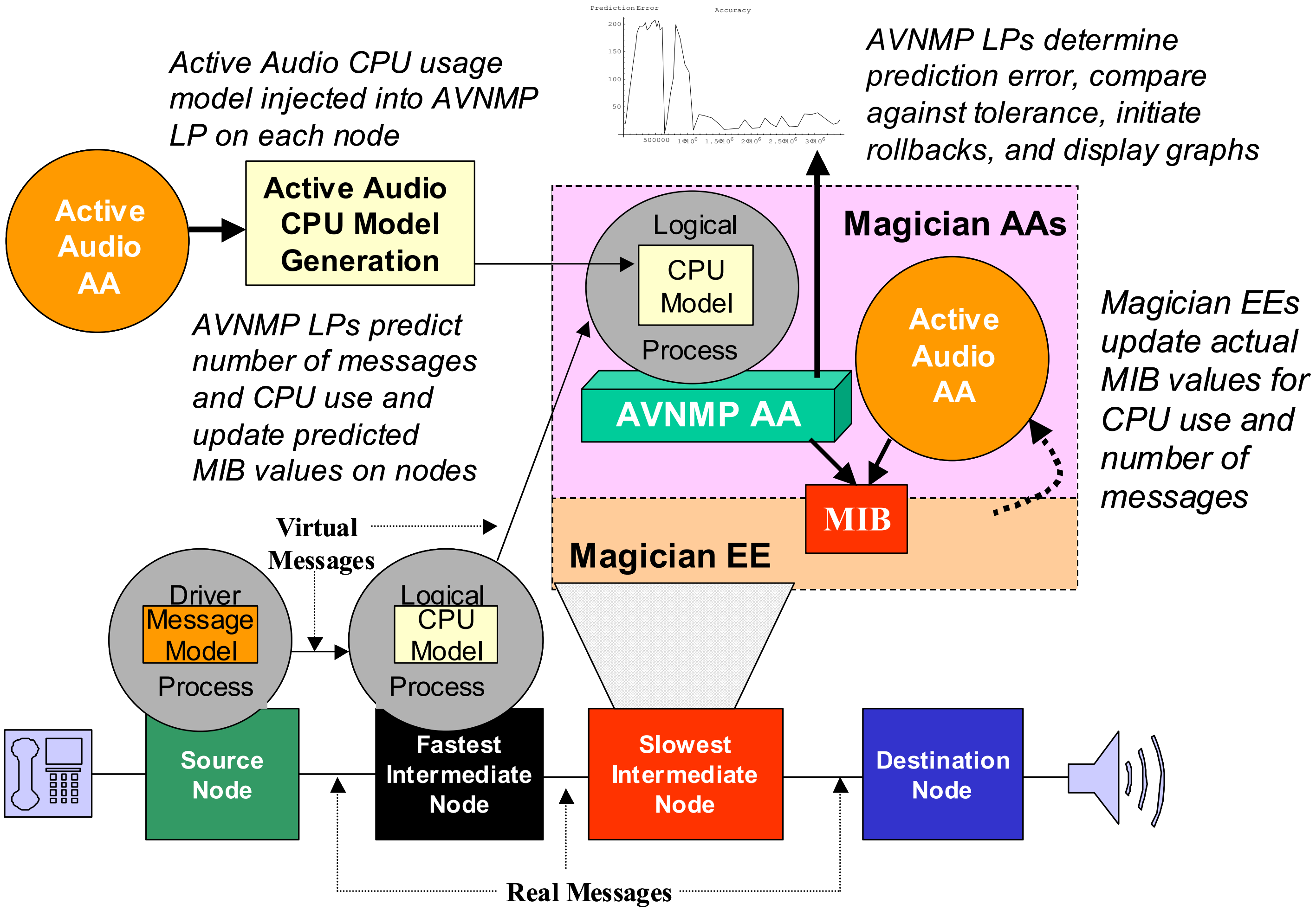
| Parameter | Value |
|---|---|
| Good Packets | 2278 |
| Malicious Packets | 379 |
| Time-to-Live (ms) | 39.87 |
| NIST Model Threshold 99th percentile | 36.35 |

Difference in Per Packet CPU Usage (TTL - NIST Model)



# *Experiment #2: Predict CPU Usage among Heterogeneous Network Nodes*

Goals: (1) Show improved look ahead into virtual time ***AND***
(2) Show fewer tolerance rollbacks in the simulation



*Active Audio CPU usage model injected into AVNMP LP on each node*

*AVNMP LPs determine prediction error, compare against tolerance, initiate rollbacks, and display graphs*

*AVNMP LPs predict number of messages and CPU use and update predicted MIB values on nodes*

*Magician EEs update actual MIB values for CPU use and number of messages*

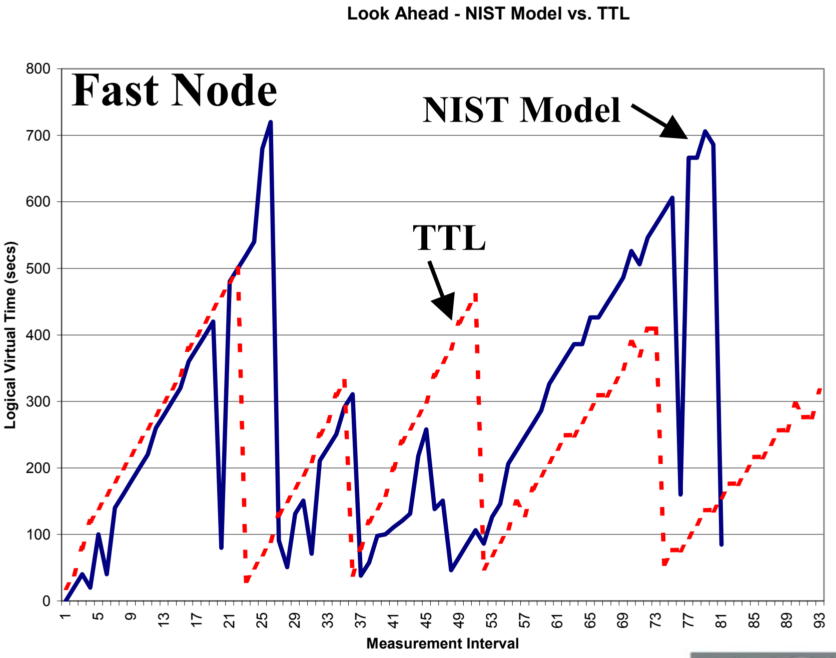
# CPU Prediction: Experiment Results

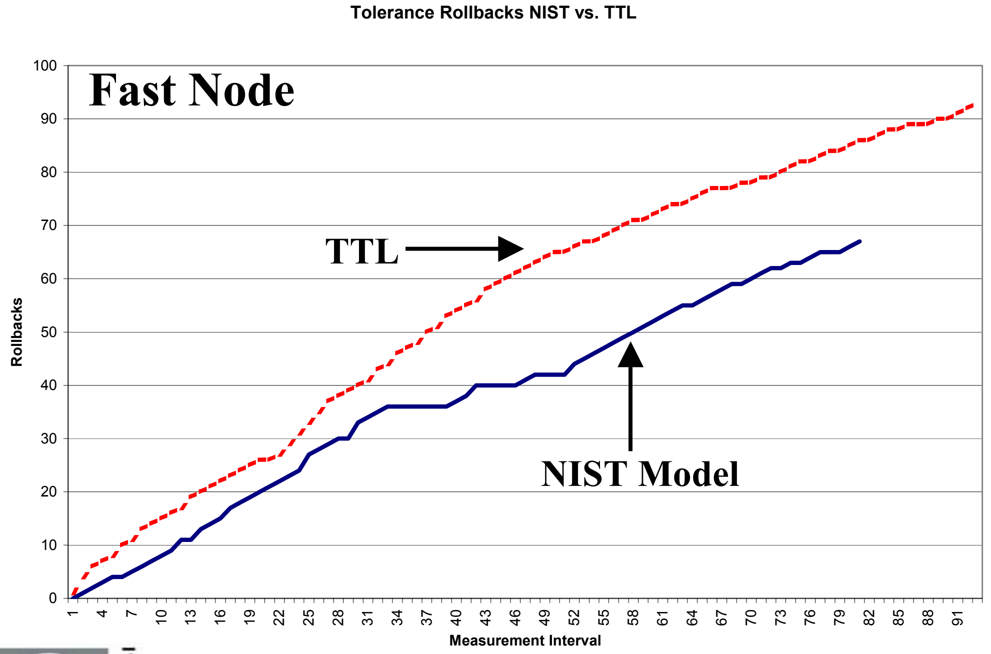
## Summary of Results

| | TTL | | NIST Model | |
|---|---|---|---|---|
| Metric | Fast Node | Slow Node | Fast Node | Slow Node |
| Maximum Look-ahead (s) | 265 | 0 | 437 | 28 |
| Tolerance Rollbacks | 93 | 47 | 67 | 20 |

## Experiment Parameters

| | TTL | | NIST Model | |
|---|---|---|---|---|
| Parameter | Fast Node | Slow Node | Fast Node | Slow Node |
| Avg. CPU Time (ms and ccs) | 7<br>2,340,750 | 7<br>693,000 | 3<br>900,000 | 16.5<br>1,633,478 |
| Error Tolerance +-10% (ccs) | 234,075 | 69,300 | 90,000 | 163,347 |
| Avg. Measurement Interval (s) | 8.8 | 12.1 | 10.1 | 7 |





# Publications Since June 2000

**Papers**

Y. Carlinet, V. Galtier, K. Mills, S. Leigh, A. Rukhin, “Calibrating an Active Network Node,” *Proceedings of the 2nd Workshop on Active Middleware Services*, ACM, August 2000.

V. Galtier, K. Mills, Y. Carlinet, S. Leigh, A. Rukhin, “Expressing Meaningful Processing Requirements among Heterogeneous Nodes in an Active Network,” *Proceedings of the 2nd International Workshop on Software Performance*, ACM, September 2000.

V. Galtier, K. Mills, Y. Carlinet, S. Bush, and A. Kulkarni, “Predicting and Controlling Resource Usage in a Heterogeneous Active Network”, accepted by *3rd International Workshop on Active Middleware Services,* ACM, August 2001.

V. Galtier, K. Mills, Y. Carlinet, S. Bush, and A. Kulkarni, “Predicting Resource Demand in Heterogeneous Active Networks”, accepted by *MILCOM 2001,* October 2001.

Papers available on the project web site: **http://w3.antd.nist.gov/active-nets/**

**Dissertation**

V. Galtier, Toward finer grain management of computational resources in heterogeneous active networks (Vers une gestion plus fine des ressources de calcul des réseaux actifs hétérogènes), Henri Poincaré University Nancy I, Advisors: André Schaff and Laurent Andrey, projected graduation October 2001.

# *Future Research (& Failures)*

- Improve Black-box Model (recent failures)
  - Space-Time Efficiency
  - Account for Node-Dependent Conditions
  - Characterize Error Bounds
- Investigate Alternate Models
  - ***White-box Model (currently underway)***
  - Lower-Complexity Analytically Tractable Models (original failure)
  - Models that Learn
- Investigate Prediction based on Competition
  - Run and Score Competing Predictors for Each Application
  - Reinforce Good Predictors
  - Use Prediction from Best Scoring Model

# Additional Work Since June 2000

## Investigating white-box approach to model CPU needs for mobile code

### Calibrate EE by Functions

| | Calibrated CPU Usage | | |
|---|---|---|---|
| EE Function | Avg. | Std Dev | 99th P |
| deliverToApp | t1 | s1 | p1 |
| getAddress | t2 | s2 | p2 |
| getCache | t3 | s3 | p3 |
| getDst | t4 | s4 | p4 |
| intValue | t5 | s5 | p5 |
| routeForNode | t6 | s6 | p6 |

### Active Application

```
Integer f = (Integer)n.getCache().get(getDst())
  if (f != null) { next = f.intValue();
        if (n.getAddress() != getDst())
           { return n.routeForNode(this, next); }
  else { return n.deliverToApp(this, dpt); }
```

### Active Application Model

```
L = delay (t3 + t4)
  if (c1) {delay (t5 + t2 + t4)
      if (c2) delay (t6)}
  else {delay (t1)}
```

**For each arriving packet, determine conditions and sum delays based on EE function calibration.**

### Some Preliminary Results (using the PLAN EE) – all times are us/packet

| Statistic | | AA1 | AA2 | AA3 | AA4 | AA5 |
|---|---|---|---|---|---|---|
| Average | Predicted | 205 | 164 | 175 | 183 | 333 |
| | Measured | 150 | 172 | 197 | 250 | 334 |
| Standard Deviation | Predicted | 163 | 61 | 122 | 240 | 96 |
| | Measured | 156 | 154 | 165 | 351 | 200 |
| 99th Percentile | Predicted | 318 | 224 | 239 | 484 | 500 |
| | Measured | 475 | 224 | 255 | 1455 | 1493 |